Estimating Energy usage of Transactions in Mobile Applications

Vijaya Shetty S[#1], Sarojadevi H[#2], Navya K.M [#3]

[1], *Department. of CSE, NMIT Bangalore, India*

[2] *Deptartment of CSE, NMAMIT Mangalore, India*

[3] *Department of CSE, NMIT Bangalore, India*

Abstract — *Performance of transactions in mobile applications is gaining importance due to the increased usage of these applications in mobile phones. The increased functionality of transactions in these applications incurs higher energy cost and results in degradation of performance in mobile phones. To improve the energy consumption of mobile application transactions, developers need detailed information about the energy consumption of transactions in their applications. This paper presents a review of different tools and techniques available for energy estimation of mobile applications. The paper also proposes a new technique which provides the code level energy estimation of transactions in mobile applications. This technique instruments the bytecode of the application to obtain execution paths of different transactions through the code, analyses the execution traces of the transactions and estimates the energy usage for each transaction and the application as a whole. The energy consumption feedback is given to the developer for further optimization of the code. The proposed technique does not require any expensive hardware for energy monitoring as it is based on bytecode instrumentation and profiling.*

Keywords — *Performance, Mobile Application, Bytecode, Energy estimation, Profiling.*

I. INTRODUCTION

The advent of smartphones allows users to carry more computational power in their hands. The usability of these devices is defined by energy consumption of mobile applications they use on their phones. Poor design of transactions in these applications can reduce the battery lifetime of mobile phones. The user reviews about the applications reveal many complaints related to energy usage of different mobile applications. Design of optimized, energy efficient transactions in mobile applications can increase user satisfaction. Many researchers have focused their work on evaluation of energy consumption and performance for mobile devices, focusing on hardware component and application code. Research in estimating the energy usage of mobile devices has explored wide variety of techniques ranging from specialized hardware, cycle accurate simulators to calibrated software based energy profiler that give a rough idea of energy estimate. The approaches that estimate the hardware component level energy usage such as disk and CPU state do not provide enough information about the energy consumed at the code level and their realisation is very expensive. Similarly cycle accurate simulators and operating system level instrumentation can slow down the overall process beyond the point of usability. Because of these limitations there are no tools available to developers to estimate software power consumption easily and accurately.

To address these limitations, a new technique is being proposed which will estimate the energy usage of transactions in mobile application at code level. The proposed technique is inexpensive as it does not require any special hardware for monitoring the energy usage of the application.

Our research is focussed on energy estimation of transactions in mobile applications. A transaction processing system is basically an information based system. A transaction in general is an exchange, usually a request and response between a user (human or software) and the system [1], [2], [3]. A transaction may require fetching of data from the database and it's processing before presenting it to the user. In this context, users are mobile users.

II. TOOLS FOR MEASURING ENERGY ESTIMATION

Android is a Linux based platform that can be used for the development of the transactional mobile applications. During development of the mobile application, it is necessary to analyse few non functional requirements like performance, energy consumption, power usage etc. Performance analysis of these applications can be done with traceview and, energy and power consumption estimation can be done with power tutor.

A. Traceview

Android software development tool kit provides tools for debugging, profiling and monitoring [4]. The dalvik debug monitor server (DDMS) belongs to this kit, contains a traceview that provides timeline and profile panel in a graphical user interface. The timeline panel provides start/stop time of each thread. The profile panel gives the summary of performance for each method and its children methods. The summary includes inclusive and exclusive execution time and the number of calls/recursive of the method. The inclusive execution time is the total time spent by a called method to execute from first instruction to last instruction. The exclusive execution time is the trace that is done either by changing application's source code to call both start and stop tracing methods or by

manually starting and stopping the trace after the application is already executed. This is very useful tool for profiling. However, it has a limitation on amount of data that it can trace, since it stores all the traced data in a buffer with limited size and, it also depends on the amount of free RAM that is available on the mobile device. If the buffer overflows, then all the data after the overflow will be lost. Thus this tool cannot be used for analysing large applications.

B. Power Tutor

Power Tutor was implemented for the android platform which is used for estimating power that is being consumed by different components like CPU, network interface, GPS and so on[5]. This application allows the developers to see the changes on power efficiency. This tool is built for specific smartphone model as they have different power dissipation behavior. With single application running in the system, this tool provides accurate results. However, with several applications running at the same time the energy estimation may diverge from real power and energy consumption of the system as whole.

C. Trepn Profiler

Trepn profiler [6] is used to profile performance and power dissipation of processors. The data is collected in real time during execution and this data can be used for offline analysis. It is designed to identify applications CPU usage, excess data use and battery use. The power dissipation information is obtained by hardware sensors that are present in these processors.

D. VisualVM

Visual VM is a visual tool; which is integrated with jdk tools and profiling capabilities. This tool is used to monitor, profile, taking thread dumps, creates report which has all the necessary information related to the monitored applications. Profiling tool [7] is used for performance analysis, debugging, to know the amount of memory used, time taken for executing particular component and the amount of CPU usage for executing the application. This tool is used by the application developers, system administrators and application users.

III. APPROACHES FOR ESTIMATING ENERGY CONSUMPTION AT SOFTWARE LEVEL

To improve energy consumption of mobile application, developers need to have a detailed behavior of energy consumption of their applications.

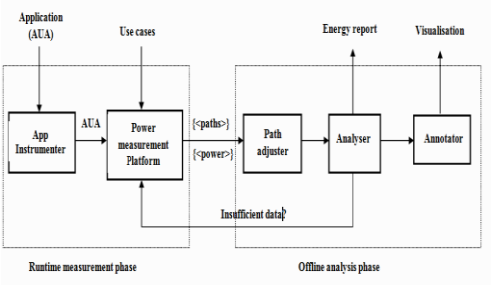

Fig. 1 Estimating source line level energy estimation

The first approach is used to calculate source line level energy consumption information by combining hardware based power measurements with program analysis [8]. There are two phases in their approach; runtime measurement phase and offline analysis as shown in Fig. 1.The runtime measurement phase has application under test and the usecases for which user wants to test energy measurements. The App Instrumenter records the path traversal information by using efficient path profiling technique proposed by larus and ball. The approach builds the control flow graph (CFG) for each method in AUA, and then each edge in the CFG is assigned with unique path ID. Later it calculates maximal spanning over the CFG and uses this information for incrementing path ID counter. The power measurement platform is based on LEAP node which is an x86 platform based on ATOM N550 processor that runs on android 3.2. .In offline analysis phase, the analyzed tuples and power samples are taken as input from the runtime analysis phase to produce energy consumption at source line. The path adjuster examines the paths in CFG and adjusts the energy measurement. The adjusted energy measurement and paths are the inputs to the analyzer. The analyzer uses robust regression technique to calculate the energy consumption at source line level. It also determines if there is enough data collected during runtime analysis phase to perform liner regression; if not analyzer will direct the tester to repeat the test and collect more data points. Finally the annotator creates the graphical representation of the energy report. The adjusted energy measurement and paths are the inputs to the analyser. The analyzer uses robust regression technique to calculate the energy consumption at source line level. It also determines if there is enough data collected during runtime analysis phase to perform liner regression; if not analyzer will direct the tester to repeat the test and collect more data points. Finally the annotator creates the graphical representation of the energy report.

Cycle accurate simulators have been developed to estimate the software energy consumption (such as sim-P analyser [9] and wattch [10]). This approach will simulate the processors action and estimate energy consumption in each cycle. Since most of the android applications take user inputs via graphical user interface it is difficult to provide accurate estimation of energy consumption.

The next approach is focused on energy consumption of the operating system at a system call level [11], [12], [13]. They build power models at system call level and this model describes the power consumed as a function of some features of the system call level, for example CPU utilization as input parameters. These models [14], [15] are then used to profile energy consumption at system level during execution of the target program.

As there are no well known tools that can accurately and easily estimate software power consumption, developers follow well-known best practices that provide general advice and guidelines. The coding practices which can improve overall performance of the mobile application[16],[17] are: the developers must not create unnecessary objects which can lead to periodic garbage collection which creates negative impact on mobile applications performance, use static final for constants, use of getters and setters methods should be avoided, use of package access instead of private access in a private inner class, use of static invocations instead of virtual invocations, avoid of accessing array length variable in the body of the loop, avoid using floating point because it is 2x times slower than integers in android applications, by using share intent it works similar to your own activity and can share the data back and forth between the applications and returns back to your activity when it is closed.

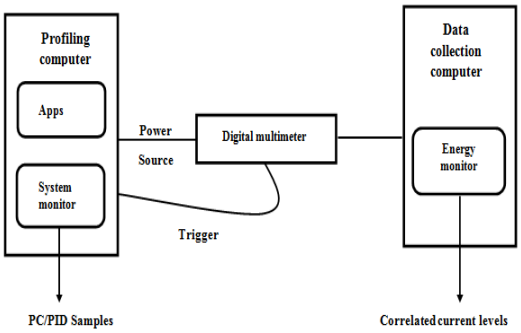

Fig. 2a Data collection phase

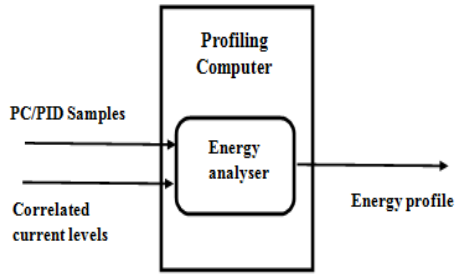

Fig. 2b Online analysis phase

Another complementary approach is Sesame [18], which generates and adapts energy models without external measurement. To gain lower overhead in the performance this tool schedules the computational task when system is idle and when it is connected to the power supply. Sesame uses CPU timings statistics, memory usage and advanced configuration and power interface which is available in modern mobile devices. However this approach doesn't provide the correct estimation of energy of one application instead it gives the measure of all the applications running concurrently.

Choice of software's algorithms and architecture has the significant effect on energy consumption of the system. Software developers lack with the tools to track the energy hungry section of the code and hence they have to rely on assumptions when they are trying to optimize the code. Eprof[19],[20] is a profiler tool that relates the energy consumption at code level. This gives energy consumed at the CPU and memory subsystem. It does not require any specialized hardware to energy profile software, it just requires minimum changes in the coding. Energy profile requires two types of information: code location which cased energy consumption and amount of energy spent by the code. Eprof implements two components for each device: observation of energy relevant activity and estimation of amount of energy consumed by that activity as shown in fig. 3. When any activity is started in the CPU Eprof records the path and code location from where it originated. The energy consumption in memory and CPU takes place while code is executing. The energy profiling for CPU and memory is done by hardware performance event counter (HPCs). Eprof allows an activity to program on HPCs, when a counter threshold is reached an interrupt is generated. The path of currently running thread is captured. Eprof is also used to profile the energy consumption of peripheral devices like hard disk, network cards etc.

Another tool for profiling energy usage of application is powerscope [21]. It relates energy consumption to program structure. Powerscope is used to determine the total amount of energy that is consumed by a particular process during certain time period and also specifies the next target in the code to be optimized so that the system can be improved and meets the design goals. This tool is a two stage process: data collection phase (Fig. 2a) and offline analysis phase (fig. 2b). During data collection phase system activity of the profiling computer and power consumption is observed. Later tool generates the energy profile during analysis phase. To sample the current drawn multimeter is used. System monitor and energy monitor are the two software components of the powerscope. The system monitor is responsible for data collection. When a system activity is triggered by the multimeter the current samples are recorded and also values of program counter and the processor identifier of currently running process are recorded. The energy monitor is responsible for collecting and storing current samples. The energy analyzer generates the energy profile of system activity. It takes raw data collected in data collection phase and associates current samples collected by energy

monitor with the samples collected by system monitor. The analyzer generates a summary of energy usage by process. Each entry contains total time spent executing the process, total energy usage of the process and average power usage. By using these data optimization can be done in the code just by changing few tens of lines to improve performance of the application.

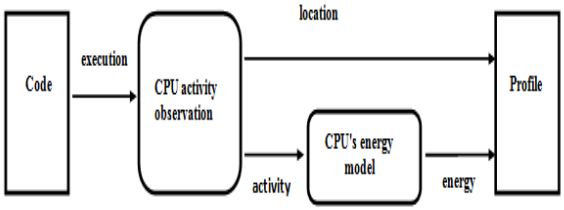

Fig. 3 Overview of Eprof

Previous studies on power modeling used external power measurement component. AppScope [22] is an android application which is an automatic online tool to estimate energy consumption for each smartphone. This tool obtains data about components types and their configuration and also detects the update rate automatically by considering update activity of battery monitoring unit. For each hardware component this tool creates a scenario to perform power analysis. The workload is assigned for component, when it triggers power state of each component is recorded.

All the above discussed approaches have drawbacks of using hardware components to estimate the energy that is consumed by the applications. These components are very expensive and they run several thousand times slower than the actual software.

To overcome this drawback, there are few major challenges that need to be solved[23]: On the energy side, software developers have to necessarily understand the energy usage of mobile application, On the performance side, the approach used to optimize performance is necessarily applicable to mobile device due to the difference in workloads and performance bottlenecks, On the Memory side : Memory must be carefully managed while creating the applications, On the Processing power: CPU's usage differs from phone to phone and this must be taken into consideration by developers. A technique based on execution traces and bytecode analysis is the solution to above mentioned problems[24]. Bytecode Profiling is used by the application developers, system administrator, application user's for performance analysis and debugging. Bytecode profiling is a technique used for solving performance related problems in various domains. It enhances the program to trace its execution, gather data, monitor memory usage etc.

To improve the performance of the system, application's source code has to be converted to its bytecode. There are three main advantages of using bytecode instead of native code. First is portability; systems with different processor architecture have different instruction sets. If the application is compiled to bytecode it can be executed in any environment that has virtual machine installed in it. Second is security; bytecode is designed to validate that it doesn't have the instructions that a malicious programmer would use to hide their assaults. Third is size: memory consumed by the bytecode is less compared to its actual source code.

IV. Proposed System

The architecture of proposed system for energy estimation of transactions in mobile applications is shown in fig. 4. Running the application's actual source code for energy estimation takes large amount of OS resources and hence this may lead to performance degradation of the overall approach. To overcome this limitation the technique of bytecode profiling is used. CPU's clock cycles and memory consumption can be reduced by profiling the bytecode rather than the source code.

The approach uses the following steps to estimate energy consumption for transactions in mobile applications.

Step 1: Initially consider the applications source code (bytecode) as input to the process.

Step 2: Instrument the application's source code(byte code) to track execution paths of different transactions in the application. Tracking information may be locations in bytecode.

Step 3: For each transaction obtain the execution traces by grouping the instructions in path of that transaction.

Step 4: Analyse the traces which is obtained for transactions and apply the energy cost values defined in the energy profile to estimate energy consumption of transactions. Algorithm to calculate the energy estimate of transactions is given in Algorithm 1.

Step 5: To find the energy estimate for the overall application, find sum of energy estimates of all transactions in the application. Algorithm to calculate energy estimate of the application is given in Algorithm 2.

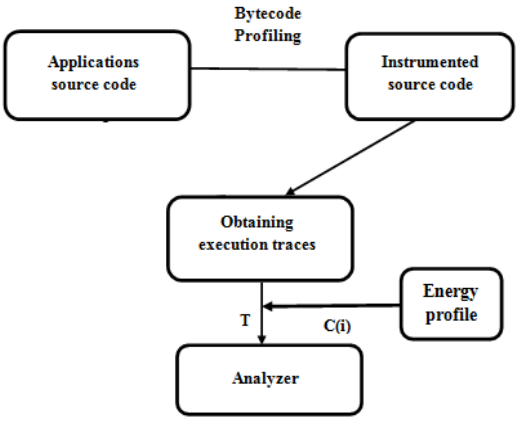

Fig. 4 Overview of proposed technique

Algorithm 1: Estimating energy consumption for Transactions

Input: I_S: Instrumented Source code(bytecode), T: Transaction, T_{trace}: Trace of T,
C(i): Energy Cost for instruction i in energy Profile.
Output: Energy estimate in joules.
For all Transaction T in I_S do
Energy consumed ← 0
T_{trace} ← Generate trace(T)
For all instruction i in T_{trace} do
$^{E}T \leftarrow \Sigma_i C(i)$
Return E_T

The energy profile C(i) defines energy cost values for each instruction i. The analyzer computes energy estimate using the trace information and energy profile data. Estimation is done at transaction level and for the whole application. To estimate energy consumption for transaction T, the energy cost in the energy profile for each instruction in the path of transaction T is summed up. Instructions in the transactions may involve connectivity to database, retrieval query and post processing of retrieved data. Finally, the analyzer computes energy estimate of whole application by summing the energy estimate values of all transactions.

Algorithm 2: Estimating energy consumption for the Application

Input: I_S: Instrumented Source code, $E_T[\ ]$: Energy Estimates of Transactions
Output: Energy estimate in joules.
For all Transactions i in E_S do
$E_A \leftarrow \Sigma_i E_T[i]$
Return E_A

V. Conclusions

This research paper presents a review of various tools and techniques available for measuring and estimating the energy consumption of mobile applications. The paper also proposes a new technique for estimating energy consumption of transactions in mobile applications via bytecode profiling. The profiling technique proposed makes profiling inline with the source code. Bytecode of the application is instrumented for profiling. No special hardware is required for monitoring the energy consumption of transactions. The system is expected to be faster in execution as the actual profiling is at bytecode level. Bytecode uses less memory and execution time. Previous approaches using bytecode profiling technique are designed for the mobile applications as a whole. Our proposed system is mainly for monitoring the energy consumption of transactions as they are the prime causes of performance degradation in mobile applications. The feedback obtained from this system can be used by the application developers to optimize transactions in the application. Our future work will be evaluation of this system on transaction based android mobile application.

Acknowledgment

Our thanks to Management, Principal and, Head of the department for their continuous research encouragement and motivating guidelines.

References

[1] Vijaya Shetty S, H. Sarojadevi, “e-Business Performance Issues, Quality Metrics and Development Frameworks”, International Journal of Computer Applications, ISSN- 0975 – 8887, pp. 42-47, Volume 55– No.7, October 2012.
[2] Vijaya Shetty S, Dr.H.Sarojadevi,”Performance Analysis of Transactional Applications in AMD Quad-Core and Intel i5 Processor Systems”, Advanced Research in Engineering and Technology, ISBN-978-81-910691-7-8, pp 381-388, Volume 8, Feb 2014.
[3] S. V. Shetty, H. Sarojadevi, B. Sriram, "A Highly Robust Proxy Enabled Overload Monitoring System (P-OMS) for E-Business Web Servers”, Smart Innovation, Systems and Technologies , ISBN: 978-81-322-2201-9 (Print) 978-81-322-2202-6 (Online), Volume 33,pp. 385-394, Dec 2014.
[4] http://developer.android.com/tools/help/traceview.html, “Traceview”.
[5] https://play.google.com/store/apps/details?id=edu.umich.PowerTutor&hl=enttp://developer.android.com/guide/components/aidl.html
[6] https://developer.qualcomm.com/software/trepn-power-profiler, “Trepn Power Profiler”.
[7] https://visualvm.java.net/, “VisualVm 1.3.8”.
[8] D. Li, S. Hao, W. G. Halfond, and R. Govindan, “Calculating source line level energy information for android applications,” in Proceedings of the 2013 International Symposium on Software Testing and Analysis (ISSTA), July 2013.
[9] T. Mudge, T. Austin, and D. Grunwald, “The Reference Manual for the Sim-Panalyzer Version 2.0.”
[10] D. Brooks, V. Tiwari, and M. Martonosi, “Wattch: A framework for architectural-level power analysis and optimizations,” in Proceedings of the 27th International Symposium on Computer Architecture (ISCA),2000.

[11] Shuai Hao, Ding Li, William G. J. Halfond, Ramesh Govindan," Estimating Mobile Application Energy Consumption using Program Analysis",IEEE,2013.
[12] T. Li and L. K. John, "Run-time modeling and estimation of operating system power consumption," ACM SIGMETRICS Performance Evaluation Review, 2003.
[13] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. Dick, Z. Mao, and L. Yang, "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphone",in Proc. of IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, pages 105–114. ACM, 2010.
[14] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. Dick, Z. Mao, and L. Yang. "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smart phones", ACM, 2010.
[15] A.Caroll and G.Heiser,"an analysis of power consumption in a smartphone",in USENIX ATC,2010.
[16] D. Li and W. G. Halfond, "An investigation into energy-saving programming practices for android smartphone app development." in Proceedings of the 3rd International Workshop on Green and Sustainable Software (GREENS), 2014.
[17] Sona Mundody, Sudarshan. K, "Evaluating the Impact of Android Best Practices on Energy Consumption" , International Conference on Information and Communication Technologies,2013.
[18] M. Dong and L. Zhong. Sesame:"Self-Constructive System Energy Modeling for Battery-Powered Mobile Systems", MobiSys, 2011.
[19] Simon Schubert, Dejan Kostic, Willy Zwaenrpoel, Kang G. Shin, " Profiling software for energy consumption", In IEEE international conference on green computing and communication,2012.
[20] Luca Ardito, Giuseppe Procaccianti, Marco Torchiano, Giuseppe Migliore," Profiling Power Consumption on Mobile Devices", Third International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies, Energy 2013.
[21] J. Flinn and M. Satyanarayanan. Powerscope: A Tool for Profiling the Energy Usage of Mobile Applications. In Second IEEE Workshop on Mobile Computing Systems and Applications, pages 2–10. IEEE, 1999.
[22] Yoon Chanmin, Kim Dongwon, Jung Wonwoo, Kang Chulkoo, Cha Hojung, "Appscope : Application energy metering framework for android smartphone using kernel activity",ACM,2012.
[23] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Where is the energy spent inside my app? Fine Grained Energy Accounting on Smart phones with Eprof", In Proc. of EuroSys, 2012.
[24] Shuai Hao, Ding Li, William G. J. Halfond, Ramesh Govindan," Estimating Android Applications CPU Energy Usage via Bytecode Profiling",IEEE,2012.